

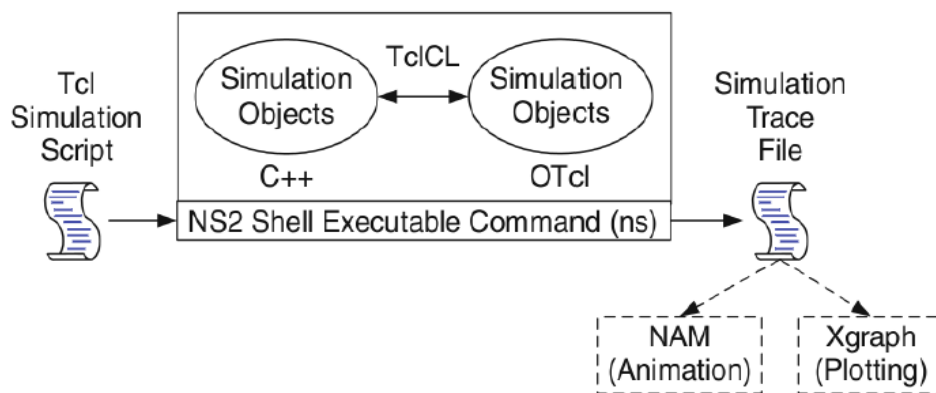
List Of Lab Experiments

1. Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.
2. Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.
3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.
4. Develop a program for error detecting code using CRC-CCITT (16- bits).
5. Develop a program to implement a sliding window protocol in the data link layer.
6. Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.
7. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.
8. Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.
9. Develop a program for a simple RSA algorithm to encrypt and decrypt the data.
10. Develop a program for congestion control using a leaky bucket algorithm.

Introduction to NS-2:

- Widely known as NS2, is simply an event driven simulation tool.
- Useful in studying the dynamic nature of communication networks
- Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.
- In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors.

Basic Architecture of NS2



Tcl scripting

- Tcl is a general purpose scripting language. [Interpreter]
- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL

Syntax: command arg1 arg2 arg3

➤ Hello World:

```
puts stdout{Hello, World!}  
Hello, World!
```

➤ Variables: Command Substitution

```
set a 5 set len [string length foobar]  
set b $a set len [expr [string length foobar] + 9]
```

➤ Simple Arithmetic: expr 7.2 / 4

➤ Procedures

```
proc Diag {a b} {  
  set c [expr sqrt($a * $a + $b * $b)]  
  return $c }  
puts —Diagonal of a 3, 4 right triangle is [Diag 3 4]
```

Output: Diagonal of a 3, 4 right triangle is 5.0

➤ Loops

```
while{$i < $n} { for {set i 0} {$i < $n} {incr i} {  
  .....  
} }
```

Initialization and Termination of TCL Script in NS-2 :

An ns simulation starts with the command

set ns [new Simulator]

Which is thus the first line in the tcl script? This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new. In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using —open command:

#Open the Trace file

set tracefile1 [open out.tr w]

\$ns trace-all \$tracefile1

#Open the NAM trace file

set namfile [open out.nam w]

\$ns namtrace-all \$namfile

The above creates a dta trace file called —**out.tr** and a nam visualization trace file called —**out.nam**.

Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called —tracefile1 and —namfile respectively. Remark that they begins with a # symbol. The second line open the file —out.tr to be used for writing, declared with the letter —w. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command \$ns flush-trace. In our case, this will be the file pointed at by the pointer —\$namfile, i.e the file —out.tr.

The termination of the program is done using a `—finish` procedure.

#Define a “finish” procedure

```
Proc finish { } {
```

```
global ns tracefile1 namfile
```

```
$ns flush-trace
```

```
Close $tracefile1
```

```
Close $namfile
```

```
Exec nam out.nam &
```

```
Exit 0
```

```
}
```

The word `proc` declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method `—flush-trace` will dump the traces on the respective files. The tcl command `—close` closes the trace files defined before and **exec** executes the `nam` program for visualization. The command **exit** will ends the application and return the number 0 as status to the system. Zero is the default for a clean exit. Other values can be used to say that is a exit because something fails. At the end of `ns` program we should call the procedure `—finish` and specify at what time the termination should occur. For example, **\$ns at 125.0 “finish”** will be used to call `—finish` at time 125sec. Indeed, the **at** method of the simulator allows us to schedule events explicitly.

The simulation can then begin using the command: **\$ns run**

Structure of Trace Files:

When tracing into an output ASCII file, the trace is organized in 12 fields as follows in fig shown below, The meaning of the fields are:

Event	Time	From	To	PKT	PKT	Flags	Fid	Src	Dest	Seq	Pkt
		Node	Node	Type	Size			Addr	Addr	Num	id

1. The first field is the event type. It is given by one of four possible symbols `r`, `+`, `-`, `d` which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.
2. The second field gives the time at which the event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.
9. This is the source address given in the form of `—node.port`.
10. This is the destination address, given in the same form.

11. This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes

12. The last field shows the Unique id of the packet.

Awk- An Advanced

awk is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers. awk is not just a command, but a programming language too. In other words, awk utility is a pattern scanning and processing language. It searches one or more files to see if they contain lines that match specified patterns and then perform associated actions, such as writing the line to the standard output or incrementing a counter each time it finds a match.

Syntax:

awk option 'selection criteria {action}' file(s)

- 1. Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.**

File: lab1.tcl

#Create a new Simulation Instance

set ns [new Simulator]

#Turn on the Trace and the animation files

set f [open out.tr w]

set nf [open out.nam w]

\$ns trace-all \$f

\$ns namtrace-all \$nf

#Define the finish procedure to perform at the end of the simulation

```
proc finish {} {  
    global f nf ns  
    $ns flush-trace  
    close $f  
    close $nf  
    exec nam out.nam &  
    exec awk -f 1.awk out.tr &  
    exit 0  
}
```

#Create the nodes

set n0 [\$ns node]

set n1 [\$ns node]

set n2 [\$ns node]

#Label the nodes

\$n0 label "TCP Source"

\$n1 label "UDP Source"

\$n2 label "Sink"

#Set the color

\$ns color 1 red

\$ns color 2 yellow

#Create the Topology

**\$ns duplex-link \$n0 \$n1 2Mb 10ms DropTail
\$ns duplex-link \$n1 \$n2 1.75Mb 20ms DropTail**

**#Attach a Queue of size N Packets between the nodes n1 n2
\$ns queue-limit \$n1 \$n2 10**

**#Make the Link Orientation
\$ns duplex-link-op \$n0 \$n1 orient right
\$ns duplex-link-op \$n1 \$n2 orient right**

**#Create a UDP Agent and attach to the node n1
set udp0 [new Agent/UDP]
\$ns attach-agent \$n1 \$udp0**

**#Create a CBR Traffic source and attach to the UDP Agent
set cbr0 [new Application/Traffic/CBR]
\$cbr0 attach-agent \$udp0**

**#Specify the Packet Size and interval
\$cbr0 set packetSize_ 500
\$cbr0 set interval_ 0.005**

**#Create a Null Agent and attach to the node n2
set null0 [new Agent/Null]
\$ns attach-agent \$n2 \$null0**

**#Connect the CBR Traffic source to the Null agent
\$ns connect \$udp0 \$null0**

**#Create a TCP agent and attach to the node n0
set tcp0 [new Agent/TCP]
\$ns attach-agent \$n0 \$tcp0**

**#Create a FTP source and attach to the TCP agent
set ftp0 [new Application/FTP]**

**#Attach the FTP source to the TCP Agent
\$ftp0 attach-agent \$tcp0**

#Create a TCPSink agent and attach to the node n2

```
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $sink
```

```
#Specify the Max file Size in Bytes
$ftp0 set maxPkts_ 1000
```

```
#Connect the TCP Agent with the TCPSink
$ns connect $tcp0 $sink
```

```
$udp0 set class_ 1
$tcp0 set class_ 2
```

```
#Schedule the Events
$ns at 0.1 "$cbr0 start"
$ns at 1.0 "$ftp0 start"
$ns at 4.0 "$ftp0 stop"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

```
File: lab1.awk
#!/usr/bin/awk -f
BEGIN{
```

```
    cbrPkt=0;
    tcpPkt=0;
}

{
    if(($1 == "d")&&($5 == "cbr")) {
        cbrPkt = cbrPkt + 1;
    }
    if(($1 == "d")&&($5 == "tcp")) {
        tcpPkt = tcpPkt + 1;
    }
}
```

```
END {
    printf "\nNo. of CBR Packets Dropped %d", cbrPkt;
    printf "\nNo. of TCP Packets Dropped %d", tcpPkt;
}
```


Steps for execution:

1. Open vi editor and type program.

2. Program name should have the extension “.tcl ” **vi lab1.tcl**

Save the program by pressing **ESC :wq** and press enter

3. Open vi editor and type awk program. Program name should have the extension “.awk ” **vi lab1.awk**

Save the program by pressing **ESC:wq** and press Enter key.

4. Run the simulation program : **ns lab1.tcl** Here “ns” indicates network simulator. We get the topology shown in the snapshot.

5. Now press the play button in the simulation window and the simulation will begins.

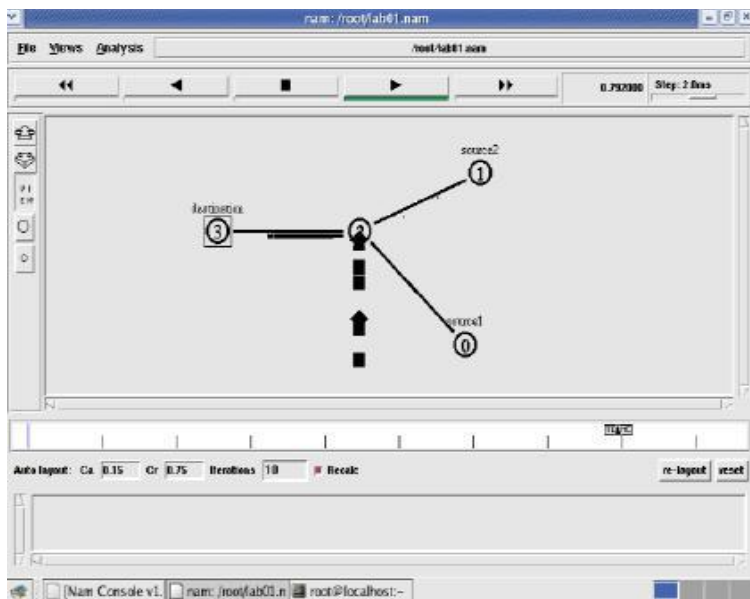
6. After simulation is completed run awk file to see the output:

awk -f lab1.awk lab1.tr

7. To see the trace file contents open the file as : **vi lab1.tr**

Output:

```
root@localho
File Edit View Terminal Tabs Help
0.1 0 2 cbr 500 ----- 0 0.0 3.0 0 0
- 0.1 0 2 cbr 500 ----- 0 0.0 3.0 0 0
r 0.10108 0 2 cbr 500 ----- 0 0.0 3.0 0 0
+ 0.10108 2 3 cbr 500 ----- 0 0.0 3.0 0 0
- 0.10108 2 3 cbr 500 ----- 0 0.0 3.0 0 0
+ 0.105 0 2 cbr 500 ----- 0 0.0 3.0 1 1
- 0.105 0 2 cbr 500 ----- 0 0.0 3.0 1 1
r 0.10608 0 2 cbr 500 ----- 0 0.0 3.0 1 1
+ 0.10608 2 3 cbr 500 ----- 0 0.0 3.0 1 1
- 0.10608 2 3 cbr 500 ----- 0 0.0 3.0 1 1
+ 0.11 0 2 cbr 500 ----- 0 0.0 3.0 2 2
- 0.11 0 2 cbr 500 ----- 0 0.0 3.0 2 2
r 0.11108 0 2 cbr 500 ----- 0 0.0 3.0 2 2
+ 0.11108 2 3 cbr 500 ----- 0 0.0 3.0 2 2
- 0.11108 2 3 cbr 500 ----- 0 0.0 3.0 2 2
+ 0.115 0 2 cbr 500 ----- 0 0.0 3.0 3 3
- 0.115 0 2 cbr 500 ----- 0 0.0 3.0 3 3
r 0.11608 0 2 cbr 500 ----- 0 0.0 3.0 3 3
+ 0.11608 2 3 cbr 500 ----- 0 0.0 3.0 3 3
- 0.11608 2 3 cbr 500 ----- 0 0.0 3.0 3 3
+ 0.12 0 2 cbr 500 ----- 0 0.0 3.0 4 4
- 0.12 0 2 cbr 500 ----- 0 0.0 3.0 4 4
r 0.12108 0 2 cbr 500 ----- 0 0.0 3.0 4 4
+ 0.12108 2 3 cbr 500 ----- 0 0.0 3.0 4 4
```



```
File Edit View Terminal Tabs Help
[root@localhost ~]# vi lab01.tcl
[root@localhost ~]# awk -f PA1.awk lab01.tr
cbr      139
cbr      143
cbr      130
cbr      149
cbr      151
cbr      154
cbr      139
cbr      159
cbr      163
cbr      145
cbr      169
cbr      171
cbr      174
cbr      177
cbr      179
cbr      182
The number of packets dropped =16
[root@localhost ~]#
```

2. Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

File : lab2.tcl

```
set ns [new Simulator]  
set f [open out.tr w]  
set nf [open out.nam w]
```

```
$ns trace-all $f  
$ns namtrace-all $nf
```

```
proc finish {} {  
    global ns f nf  
    $ns flush-trace  
    close $f  
    close $nf  
    exec nam out.nam &  
    exit 0  
}
```

```
set n0 [$ns node]  
set n1 [$ns node]  
set n2 [$ns node]  
set n3 [$ns node]  
set n4 [$ns node]  
set n5 [$ns node]
```

```
$n0 label "ping0"  
$n1 label "ping1"  
$n2 label "R1"  
$n3 label "R2"  
$n4 label "ping4"  
$n5 label "ping5"
```

```
$ns color 1 red  
$ns color 2 blue  
$ns color 3 green  
$ns color 4 orange
```

```
$ns duplex-link $n0 $n2 1Mb 10ms DropTail  
$ns duplex-link $n1 $n2 1Mb 10ms DropTail  
$ns duplex-link $n2 $n3 0.4Mb 30ms DropTail  
$ns duplex-link $n3 $n4 1Mb 10ms DropTail  
$ns duplex-link $n3 $n5 1Mb 10ms DropTail
```

```
set ping0 [new Agent/Ping]  
$ns attach-agent $n0 $ping0
```

```
set ping1 [new Agent/Ping]  
$ns attach-agent $n1 $ping1
```

```
set ping4 [new Agent/Ping]  
$ns attach-agent $n4 $ping4
```

```
set ping5 [new Agent/Ping]  
$ns attach-agent $n5 $ping5
```

```
$ns connect $ping0 $ping4  
$ns connect $ping1 $ping5
```


```
proc sendPingPacket {} {  
global ns ping0 ping1  
set intervalTime 0.001  
set now [$ns now]  
$ns at [expr $now + $intervalTime] "$ping0 send"  
$ns at [expr $now + $intervalTime] "$ping1 send"  
$ns at [expr $now + $intervalTime] "sendPingPacket"  
}
```

```
Agent/Ping instproc recv {from rtt} {  
global seq  
$self instvar node_  
puts "The node [$node_ id] received an ACK from the node $from with RTT $rtt ms"  
}
```

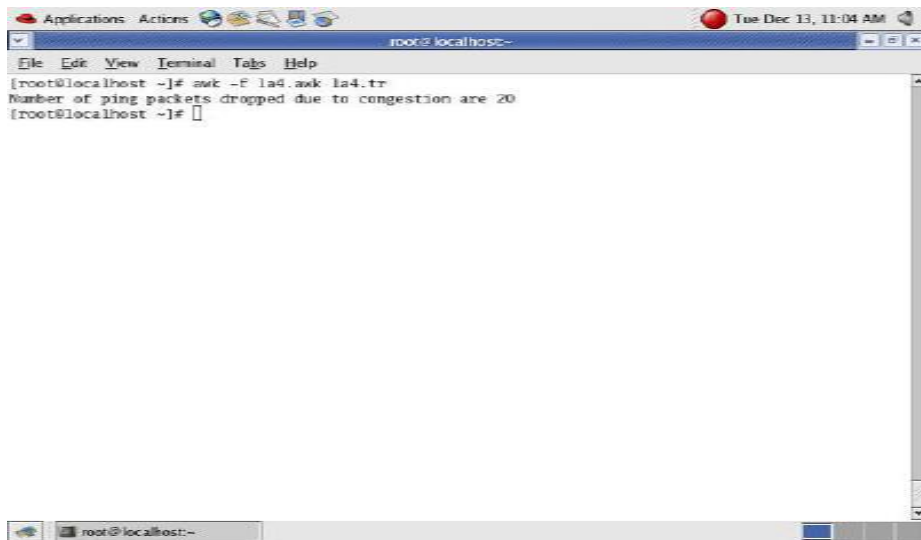
```
$ping0 set class_ 1  
$ping1 set class_ 2  
$ping4 set class_ 4  
$ping5 set class_ 5
```

\$ns run

```
printf("Total number of %s packets dropped due to congestion =%d\n",$5,drop);
```



The screenshot shows a Windows XP desktop environment. At the top, the taskbar includes the Start button, a search bar, and several application icons. The system clock in the top right corner displays 'Tue Dec 13, 10:41 AM'. The main window is a terminal application titled 'root@localhost:~'. The terminal's menu bar includes 'File', 'Edit', 'View', 'Terminal', 'Tabs', and 'Help'. The terminal window contains a continuous stream of network-related messages, such as 'node 0 received answer from 5 with round trip time 72.1 msec' and 'node 2 received answer from 3 with round trip time 88.1 msec'. At the bottom of the terminal window, the prompt '[root@localhost ~]#' is visible. Below the terminal window, a portion of another taskbar is visible, showing icons for a file explorer, a terminal window titled 'root@localhost:~', a command prompt window titled 'cmd: lan\lan', and a 'New Console' button.



3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

File : lab3.tcl

```
#set up a new instance of Simulator  
set ns [new Simulator]
```

```
#Open the trace file and animation file  
set f [open 5.tr w]  
set nf [open 5.nam w]
```

```
$ns trace-all $f  
$ns namtrace-all $nf
```

```
#Define the finish Procedure  
proc finish {} {  
    global ns f nf outFile1 outFile2  
    $ns flush-trace  
    close $f  
    close $nf  
    exec nam 5.nam &  
    exec xgraph Congestion1.xg -geometry 400x400 &  
    exec xgraph Congestion2.xg -geometry 400x400 &  
    exit 0  
}
```

```
$ns color 1 red  
$ns color 2 green
```

```
#set up the nodes  
set n0 [$ns node]  
set n1 [$ns node]  
set n2 [$ns node]  
set n3 [$ns node]  
set n4 [$ns node]  
set n5 [$ns node]  
set n6 [$ns node]  
set n7 [$ns node]
```

```
#Label the nodes  
$n0 label "TCP FTP Source"  
$n3 label "Sink Destination"  
$n5 label "TCP Telnet Source"  
$n7 label "Sink Destination"
```

```
#Create the LAN topology  
$ns make-lan "$n0 $n1 $n2 $n3 $n4 $n5 $n6 $n7" 10Mb 30ms LL Queue/DropTail  
Mac/802_3
```

```
#Set up the TCP Agents  
set tcp1 [new Agent/TCP]  
$ns attach-agent $n0 $tcp1  
set ftp1 [new Application/FTP]  
$ftp1 attach-agent $tcp1  
set sink1 [new Agent/TCPSink]  
$ns attach-agent $n3 $sink1  
$ns connect $tcp1 $sink1  
$tcp1 set class_ 1
```

```
#Set up the telnet  
set tcp2 [new Agent/TCP]  
$ns attach-agent $n5 $tcp2  
set telnet1 [new Application/FTP]  
$telnet1 attach-agent $tcp2  
set sink2 [new Agent/TCPSink]  
$ns attach-agent $n7 $sink2  
$ns connect $tcp2 $sink2  
$telnet1 set type_ $sink2  
$tcp2 set class_ 2
```

```
set outFile1 [open Congestion1.xg w]  
set outFile2 [open Congestion2.xg w]
```

```
puts $outFile1 "TitleText: Congestion Window Plot for TCP1"  
puts $outFile1 "XUnitText: SimulationTime(Secs)"  
puts $outFile1 "YUnitText: CongestionWindowSize"  
puts $outFile2 "TitleText: Congestion Window Plot for TCP2"
```

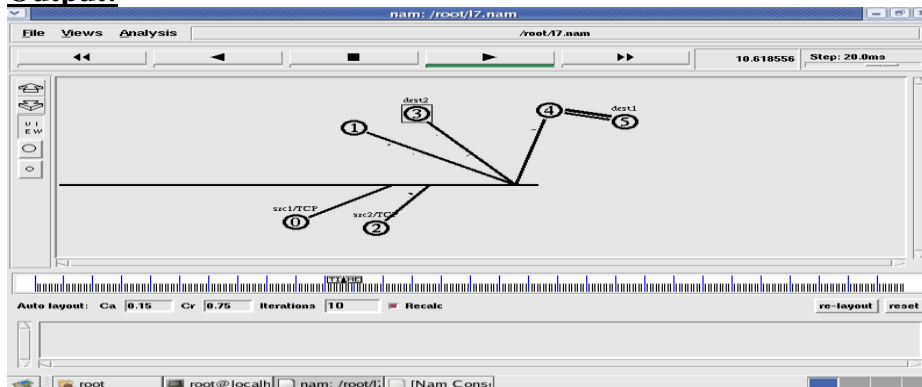


```
puts $outFile2 "XUnitText: SimulationTime(Secs)"
puts $outFile2 "YUnitText: CongestionWindowSize"
```

```
#define findWindowSize
proc findWindowSize {tcpSource outFile} {
    global ns
    set now [$ns now]
    set cWindowSize [$tcpSource set cwnd_]
    puts $outFile "$now $cWindowSize"
    $ns at [expr $now + 0.1] "findWindowSize $tcpSource $outFile"
}
}
```

```
#schedule the events
$ns at 0.0 "findWindowSize $tcp1 $outFile1"
$ns at 0.1 "findWindowSize $tcp2 $outFile2"
$ns at 0.3 "$ftp1 start"
$ns at 0.5 "$telnet1 start"
$ns at 50.0 "$ftp1 stop"
$ns at 50.0 "$telnet1 stop"
$ns at 50.0 "finish"
$ns run
```

Output:



Implement the following in Java:

4. Write a program for error detecting code using CRC-CCITT (16- bits)

Program:

```
import java.io.*;
class crc
{
    public static void main(String a[]) throws IOException
    {

        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        int[] message;
        int[] gen;
        int[] app_message;
        int[] rem;
        int[] trans_message;
        int message_bits, gen_bits, total_bits;

        System.out.println("\n Enter number of bits in message : ");
        message_bits=Integer.parseInt(br.readLine());
        message=new int[message_bits];
        System.out.println("\n Enter message bits : ");
        for(int i=0; i<message_bits; i++)
            message[i]=Integer.parseInt(br.readLine());
        System.out.println("\n Enter number of bits in gen : ");
        gen_bits=Integer.parseInt(br.readLine());

        gen=new int[gen_bits];
        System.out.println("\n Enter gen bits : ");
        for(int i=0; i<gen_bits; i++)
        {
            gen[i]=Integer.parseInt(br.readLine());
        }
        total_bits=message_bits+gen_bits-1;
        app_message=new int[total_bits];
        rem=new int[total_bits];
        trans_message=new int[total_bits];
```

```
for(int i=0;i<message.length;i++)
{
app_message[i]=message[i];
}
```

```
System.out.print("\n Message bits are : ");
for(int i=0; i< message_bits; i++)
{
System.out.print(message[i]);
}
```

```
System.out.print("\n Generators bits are : ");
for(int i=0; i< gen_bits; i++)
{
System.out.print(gen[i]);
}
```

```
System.out.print("\n Appended message is : ");
for(int i=0; i< app_message.length; i++)
{
System.out.print(app_message[i]);
}
```

```
for(int j=0; j<app_message.length; j++)
{
rem[j] = app_message[j];
}
```

```
rem=computecrc(app_message, gen, rem);
```

```
for(int i=0;i<app_message.length;i++)
{
trans_message[i]=(app_message[i]^rem[i]);
}
```

```
System.out.println("\n Transmitted message from the transmitter is : ");
for(int i=0;i<trans_message.length;i++)
{
System.out.print(trans_message[i]);
}
```

```
}
```

```
System.out.println("\n Enter received message of "+total_bits+" bits at receiver end : ");
```

```
for(int i=0; i<trans_message.length; i++)
```

```
{
```

```
trans_message[i]=Integer.parseInt(br.readLine());
```

```
}
```

```
System.out.println("\n Received message is :");
```

```
for(int i=0; i< trans_message.length; i++)
```

```
{
```

```
System.out.print(trans_message[i]);
```

```
}
```

```
for(int j=0; j<trans_message.length; j++)
```

```
{
```

```
rem[j] = trans_message[j];
```

```
}
```

```
rem=computecrc(trans_message, gen, rem);
```

```
for(int i=0; i< rem.length; i++)
```

```
{
```

```
if(rem[i]!=0)
```

```
{
```

```
System.out.println("\n There is Error in the received message!!!");
```

```
break;
```

```
}
```

```
if(i==rem.length-1)
```

```
{
```

```
System.out.println("\n There is No Error in the received message!!!");
```

```
}
```

```
}
```

```
}
```

```
static int[] computecrc(int app_message[],int gen[], int rem[])
```

```
{
```

```
int current=0;
```

```
while(true)
```

```
{
```

```
for(int i=0;i<gen.length;i++)
```

```
{
```

```
rem[current+i]=(rem[current+i]^gen[i]);
```

```

}
while(rem[current]==0 && current!=rem.length-1)
{
current++;
}
if((rem.length-current)<gen.length)
{
break;
}
}
return rem;
}
}

```

OUTPUT:Enter the number of bits in message:

5

Enter message bits:

1

0

1

1

0

Enter number of bits in gen:

3

Enter gen bits:

1

0

1

Message bits are:10110

Generator bits are:101

Appended message is:1011000

Transmitted message from the trasmitter is:

1011010

Enter received message of 7 bits at receiver end:

1

0

1

1

0

1

1

Received message is :

1011011

There is error in the received message!!!

6. Write a program to find the shortest path between vertices using bellman-ford algorithm.

PROGRAM:

```
import java.util.Scanner;
public class BellmanFord
{
    private int distances[];
    private int numberofvertices;
    public static final int MAX_VALUE = 999;
    public BellmanFord(int numberofvertices)
    {
        this.numberofvertices = numberofvertices;
        distances = new int[numberofvertices + 1];
    }
    public void BellmanFordEvaluation(int source, int destination,
        int adjacencymatrix[][])
    {
        for (int node = 1; node <= numberofvertices; node++)
        {
            distances[node] = MAX_VALUE;
        }
        distances[source] = 0;
        for (int node = 1; node <= numberofvertices - 1; node++)
        {
            for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
            {
                for (int destinationnode = 1; destinationnode <= numberofvertices;
destinationnode++)
                {
                    if      (adjacencymatrix[sourcenode][destinationnode]      !=
MAX_VALUE)
                    {
                        if (distances[destinationnode] > distances[sourcenode]
                            + adjacencymatrix[sourcenode][destinationnode])
                            distances[destinationnode] = distances[sourcenode]
                                + adjacencymatrix[sourcenode][destinationnode];
                    }
                }
            }
        }
    }
}
```

```

    }
}
for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
{
    for (int destinationnode = 1; destinationnode <= numberofvertices;
destinationnode++)
    {
        if (adjacencymatrix[sourcenode][destinationnode] != MAX_VALUE)
        {
            if (distances[destinationnode] > distances[sourcenode]
                + adjacencymatrix[sourcenode][destinationnode])
                System.out.println("The Graph contains negative egde cycle");
        }
    }
    for (int vertex = 1; vertex <= numberofvertices; vertex++)
    {
        if (vertex == destination)
            System.out.println("distance of source " + source + " to " + vertex + " is " +
distances[vertex]);
    }
}

public static void main(String... arg)
{
    int numberofvertices = 0;
    int source, destination;
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the number of vertices");
    numberofvertices = scanner.nextInt();
    int adjacencymatrix[][] = new int[numberofvertices + 1][numberofvertices
+ 1];

    System.out.println("Enter the adjacency matrix");
    for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)
    {
        for (int destinationnode = 1; destinationnode <= numberofvertices;
destinationnode++)
        {
            adjacencymatrix[sourcenode][destinationnode] = scanner
                .nextInt();
            if (sourcenode == destinationnode)
            {

```



```

        adjacencymatrix[sourcenode][destinationnode] = 0;
        continue;
    }
    if (adjacencymatrix[sourcenode][destinationnode] == 0)
    {
        adjacencymatrix[sourcenode][destinationnode] = MAX_VALUE;
    }
}
}
System.out.println("Enter the source vertex");
source = scanner.nextInt();
System.out.println("Enter the destination vertex: ");
destination = scanner.nextInt();
BellmanFord bellmanford = new BellmanFord(numberofvertices);
bellmanford.BellmanFordEvaluation(source, destination,
adjacencymatrix);
scanner.close();
}
}

```

OUTPUT: Enter the number of vertices 5

Enter the adjacency matrix

0 999 6 3 999

3 0 999 999 999

999 999 0 2 999

999 1 1 0 999

999 4 999 2 0

Enter the source vertex

5

Enter the destination vertex

1

Distance of source 5 to 1 is 6

7. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present. Implement the above program using as message queues or FIFOs as IPC channels.

Client side:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.InputStreamReader;
import java.io.DataOutputStream;
//import java.io.OutputStreamWriter;
//import java.net.InetAddress;
import java.net.Socket;
import java.io.File;
import java.io.FileOutputStream;
import java.util.Scanner;
class client
{
public static void main(String args[])throws Exception
{
String address="";
Scanner sc= new Scanner(System.in);
System.out.println("eneter server address");
address=sc.nextLine();
Socket s=new Socket(address,5000);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("snd get to start");
String str="",filename="";
try
{
while(!str.equals("start"))
str=br.readLine();
dout.writeUTF(str);
dout.flush();
filename=din.readUTF();
System.out.println("receiveing file:"+filename);
filename="client"+filename;
System.out.println("saving as file"+filename);
```

```
long sz=Long.parseLong(din.readUTF());
System.out.println("filesize:"+(sz/(1024))+"KB");
byte b[]=new byte[1024];
System.out.println("receiving file");
FileOutputStream fos=new FileOutputStream(new File(filename),true);
long bytesRead;
do
{
bytesRead=din.read(b,0,b.length);
fos.write(b,0,b.length);
}
while(!(bytesRead<1024));
System.out.println("completed");
fos.close();
dout.close();
s.close();
}
catch(Exception e)
{
//do nothing;
}
}
}
```

Server side:

```
import java.io.InputStreamReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;
import java.io.File;
import java.io.FileInputStream;
import java.net.ServerSocket;
import java.util.Scanner;
class Server
{
public static void main(String args[])throws Exception
{
String filename;
System.out.println("enter filename");
Scanner sc= new Scanner(System.in);
filename=sc.nextLine();
sc.close();
while(true)
{
ServerSocket ss =new ServerSocket(5000);
System.out.println("wait fr req");
Socket s=ss.accept();
System.out.println("connected with" +s.getInetAddress().toString());
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
try
{
String str="";
str=din.readUTF();
System.out.println("send get tok ");
if(!str.equals("stop"))
{
System.out.println("sendinf file:"+filename);
dout.writeUTF(filename);
dout.flush();
File f=new File(filename);
FileInputStream fin=new FileInputStream(f);
long sz=(int)f.length();
byte b[]=new byte[1024];
```

```

int read;
dout.writeUTF(Long.toString(sz));
dout.flush();
System.out.println("size:"+sz);
System.out.println("buffer size:"+ss.getReceiveBufferSize());
while((read=fin.read(b))!=-1)
{
dout.write(b,0,read);
dout.flush();
}
fin.close();
System.out.println("ok");
dout.flush();
}
dout.writeUTF("stop");
System.out.println("sendi complete");
dout.flush();
}
catch(Exception e)
{
e.printStackTrace();
System.out.println("erroe occuerrd");
}
din.close();
s.close();
ss.close();
}
}
}

```

Output:

- 1.Open two terminals and create files client.java and Server.java**
- 2.Open a terminal and run the server program and provide the filename to send**
- 3.Open one more terminal,run the client programand provide the IP address of the server(Loopback address)**
- 4.Type “start” at the client side.**

Server side:

Client side:

8. Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.

PROGRAM:

```
import java.net.*;
class datagram {
public static int serverPort = 666;
public static int clientPort = 999;
public static int buffer_size = 1024;
public static DatagramSocket ds;
public static byte buffer[] = new byte[buffer_size];
public static void TheServer() throws Exception {
int pos=0;
while (true) {
int c = System.in.read();
switch(c) {
case -1:
System.out.println("Server Quits.");
return;
case '\r':
break;
case '\n':
ds.send(new DatagramPacket(buffer,pos,InetAddress.getLocalHost(),clientPort));
pos=0;
break;
default:
buffer[pos++] = (byte) c;
}
}
}
public static void TheClient() throws Exception {
while(true) {
DatagramPacket p = new DatagramPacket(buffer, buffer.length);
ds.receive(p);
System.out.println(new String(p.getData(), 0, p.getLength()));
}
}
```

```
public static void main(String args[]) throws Exception {  
if(args.length == 1) {  
ds = new DatagramSocket(serverPort);  
TheServer();  
} else {  
ds = new DatagramSocket(clientPort);  
TheClient();  
}  
}  
}
```

OUTPUT:

Open 2 terminals

In 1st terminal : javac datagram.java

Java datagram

In 2nd terminal :javac datagram.java

Java datagram 1

Server side

Client side: Displaying the contents

9. Write a program for simple RSA algorithm to encrypt and decrypt the data.

PROGRAM:

```
import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;
import java.util.*;

public class encrypt
{
    private BigInteger p;
    private BigInteger q;
    private BigInteger N;
    private BigInteger phi;
private BigInteger e;
    private BigInteger d;
    private int    bitlength = 1024;
    private Random  r;

    public encrypt()
    {
        r = new Random();
        p = BigInteger.probablePrime(bitlength, r);
        q = BigInteger.probablePrime(bitlength, r);
        N = p.multiply(q);
        phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        e = BigInteger.probablePrime(bitlength / 2, r);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi)
< 0)
        {
            e.add(BigInteger.ONE);
        }
        d = e.modInverse(phi);
    }

    public encrypt(BigInteger e, BigInteger d, BigInteger N)
    {
        this.e = e;
    }
}
```



```
    this.d = d;  
    this.N = N;  
}
```

```
@SuppressWarnings("deprecation")  
public static void main(String[] args) throws IOException  
{  
    encrypt encrypt = new encrypt();  
    DataInputStream in = new DataInputStream(System.in);  
    String teststring;  
    System.out.println("Enter the plain text:");  
    teststring = in.readLine();  
    System.out.println("Encrypting String: " + teststring);  
    System.out.println("String in Bytes: "  
        + bytesToString(teststring.getBytes()));  
    // encrypt  
    byte[] encrypted = encrypt.encrypt(teststring.getBytes());  
    // decrypt  
    byte[] decrypted = encrypt.decrypt(encrypted);  
    System.out.println("Decrypting Bytes: " + bytesToString(decrypted));  
    System.out.println("Decrypted String: " + new String(decrypted));  
}
```

```
private static String bytesToString(byte[] encrypted)  
{  
    String test = "";  
    for (byte b : encrypted)  
    {  
        test += Byte.toString(b);  
    }  
    return test;  
}
```

```
// Encrypt message  
public byte[] encrypt(byte[] message)  
{  
    return (new BigInteger(message)).modPow(e, N).toByteArray();  
}
```

```
// Decrypt message
```

```
public byte[] decrypt(byte[] message)
{
    return (new BigInteger(message)).modPow(d, N).toByteArray();
}
```

OUTPUT:

Enter the plain text:

hi

Encrypting String:hi

String in Bytes: 104105

Decrypting Bytes: 104105

Decrypting String: hi

10. Write a program for congestion control using leaky bucket algorithm.

PROGRAM:

```
import java.util.*;
public class leaky
{
    public static void main(String[] args)
    {
        Scanner my = new Scanner(System.in);
        int no_groups,bucket_size;
        System.out.print("\n Enter the bucket size : \t");
        bucket_size = my.nextInt();
        System.out.print("\n Enter the no of groups : \t");
        no_groups = my.nextInt();
        int no_packets[] = new int[no_groups];
        int in_bw[] = new int[no_groups];
        int out_bw,reqd_bw=0,tot_packets=0;
        for(int i=0;i<no_groups;i++)
        {
            System.out.print("\n Enter the no of packets for group " + (i+1) + "\t");
            no_packets[i] = my.nextInt();
            System.out.print("\n Enter the input bandwidth for the group " + (i+1) + "\t");
            in_bw[i] = my.nextInt();
            if((tot_packets+no_packets[i])<=bucket_size)
            {
                tot_packets += no_packets[i];
            }
            else
            {
                do
                {
                    System.out.println(" Bucket Overflow ");
                    System.out.println(" Enter value less than " + (bucket_size-tot_packets));
                    no_packets[i] = my.nextInt();
                }while((tot_packets+no_packets[i])>bucket_size);
                tot_packets += no_packets[i];
            }
            reqd_bw += (no_packets[i]*in_bw[i]);
        }
        System.out.println("\nThe total required bandwidth is " + reqd_bw);
    }
}
```

```

System.out.println("Enter the output bandwidth ");
out_bw = my.nextInt();
int temp=reqd_bw;
int rem_pkts = tot_packets;
while((out_bw<=temp)&&(rem_pkts>0))
{
    System.out.println("Data Sent \n" + (--rem_pkts) + " packets remaining");
    System.out.println("Remaining Bandwidth " + (temp -= out_bw));
    if((out_bw>temp)&&(rem_pkts>0))
        System.out.println(rem_pkts + " packet(s) discarded due to insufficient
bandwidth");
}
}

```

OUTPUT:Enter the bucket size:5

Enter the no of groups:1

Enter the no of packets for group1:5

Enter the input bandwidth for the group1:6

The total required bandwidth is 30

Enter the output bandwidth

25

Data sent

4 packets remaining

Remaining Bandwidth: 5

4 packets discarded due to insufficient bandwidth